Challenge: K-Dimensional tree

1 Preliminaries

1.1 Research *(40 pts)*

KD-Tree, or k-dimensional tree, is a space-partitioning data structure used to organize points in a k-dimensional space. KD-Trees are particularly useful in applications involving multidimensional search keys, such as range searches and nearest neighbor searches. This data structure was introduced by Jon Bentley in 1975 primarily for range search and nearest neighbor search problems.

Do your research about KD-Trees and answer the following questions (with explanation):

1. What is the process of constructing a KD-Tree for a given set of points in 2D space? Explain how the choice of axis and split points affects the structure of the tree.
2. Discuss the time complexity of searching for a given point in a KD-Tree. How does the tree's balance affect this complexity?
3. Explain how insertions and deletions are handled in a KD-Tree. What strategies might be used to rebalance the tree if necessary?
4. How does the dimensionality of the data affect the performance of KD-Trees? Discuss the concept of the 'curse of dimensionality' in this context
5. Compare the implementation of KD-Trees using arrays versus linked lists. Discuss the advantages and disadvantages of each implementation method in terms of memory usage, access time, and ease of operations such as insertions and deletions
6. Evaluate the use of KD-Trees versus Binary Search Trees (BSTs) for multidimensional data handling. How do the structures compare when used for common operations like search, insert, and delete in a dataset with multiple attributes?

1.2 Programming *(60 pts)* You are requested to implement a KD-Tree to efficiently manage a collection of geographic locations from the "SimpleMaps World Cities" dataset. Your program should enable the insertion of city data into the tree, conduct nearest neighbor searches, and perform range queries to identify all cities within a specified rectangular geographic boundary.

Your implementation should meet the following detailed requirements:

1. ***(10 pts)* Prepare the Dataset**
   - Read the CSV file in a C++ program. You can download this dataset via https://gist.github.com/mchoi2000/e5e0486c74abdbb624db43d7f0783255
   - Extract essential information including latitude, longitude, and city names from the dataset.
   - Preprocess the data to enhance processing efficiency, such as filtering out unnecessary columns or cleaning data entries.
2. ***(25 pts)* Implement a KD-Tree**
   - **Insertion:** Insert city data into the KD-Tree, alternating between latitude and longitude for the splitting criteria based on the node depth.
   - **Range Search:** Develop a function that accepts a rectangular region defined by the bottom-left and top-right corners, both specified in terms of latitude and longitude. The function should return a list of all cities within this region.
   - **Nearest Neighbor Search:** Implement a function to locate the nearest city to a given set of geographic coordinates, optimizing the search process to minimize unnecessary subtree explorations.
   - ***Note on Distance Calculation:*** Both the range search and nearest neighbor functions should calculate distances using the Haversine formula. This formula is crucial for accurate distance measurements on the spherical surface of the Earth. For more detailed information on the Haversine formula and its implementation, refer to Haversine formula.
3. ***(25 pts)* User Interface**

- Develop a simple command-line interface that allows users to:
  - Load the list of cities from a CSV file.
  - Insert a new city into the KD-Tree directly via the command line.
  - Insert multiple new cities into the KD-Tree from a specified CSV file path.
  - Conduct a nearest neighbor search by providing latitude and longitude coordinates.
  - Query all cities within a specified rectangular region defined by its geographical boundaries.
- Ensure the program provides options for the user to output results to both the command line and a CSV file.

4. **Extended Functionality** ***(20 pts)***

- **Storing the KD-Tree Structure:** Develop functionality to serialize the KD-Tree into a file format. This will allow the tree to be saved to disk and reloaded as needed, preserving the tree structure across different sessions or for sharing among different applications.
  - Implement serialization methods to convert the tree nodes into a storable format (e.g., JSON, XML, or a binary format, etc).
  - Provide functionality for deserialization to reconstruct the KD-Tree from the stored files.
  - Provide a justification for the selected file format. Discuss the advantages and disadvantages of your chosen format in the context of serializing and deserializing the KD-Tree

2 Group registration and Submission regulations

2.1 Group Registration

- Each project group must consist of 3-4 students. No exceptions will be made to this rule.
- Group ID is generated by concatenating member's Student ID in ascending order via en dash symbol.

  Example:

  – Given the student IDs: *23127666; 23127888;23127999*.

  → **Generated Group ID**: *23127666 - 23127888 - 23127999*.

2.2 Submission regulations

- The submission file must be in the following format: [**Group_ID.rar**] or [**Group_ID.zip**], is the compression of the [**Group_ID**] folder. This folder contains:
  - The report file must be presented as a document [**Group_ID.pdf**]. This file presented research answers from **1.1** and the information of code fragment (data structures, algorithms, functions) from **1.2**.
    - If your submission is a slideshow, there must be explanation in the *Note* part of each slide.
    - Information (Names, Student IDs) must be declared clearly on the first page (or first slide) of your report. Your working progress (Which option did you choose? How much work have you completed?) should be demonstrated on this page, too.
    - The report file should be **structured, logical, clear** and **coherent**. The length of the submission should not exceed 15 pages for the document file, and 30 pages for the presentation slide.
    - All links and books related to your submission must be mentioned.
  - The programming file must be put into a [**Group_ID**] folder. The code fragment must be clear, logical and commented.

- Submission with wrong regulation will result in a "0" (zero).
- Plagiarism and Cheating will result in a "0" (zero) for the entire course and will be subject to appropriate referral to the Management Board of the CLC program for further action.